Programação Orientada a Objetos

Nádia Félix, Juliana Félix, Dirson Campos

- nadia@inf.ufg.br
- julianafelix@ufg.br
- dirson_campos@ufg.br

Programação Orientada a Objetos

- JAVA.IO
- Trabalhando com Arquivos

Programação Orientada a Objetos

Arquivos

- Assim como todo o resto das bibliotecas em Java, a parte de controle de entrada e saída de dados (conhecido como io) é orientada a objetos e usa os principais conceitos mostrados até agora: **interfaces, classes abstratas e polimorfismo.**

Programação Orientada a Objetos

Arquivos

- Muitas vezes as aplicações necessitam buscar informações de uma fonte externa ou enviar dados para um destino externo.
- Importante: não importa de onde a informação vem nem para onde está indo; também não importa qual o tipo de dado que está sendo lido ou escrito, os algoritmos de leitura e escrita, são sempre os mesmos.
- O java considera cada arquivo como um fluxo sequencial de bytes.

Programação Orientada a Objetos

Arquivos

- A interação de um programa com um dispositivo através de arquivos passa por três etapas:
  - abertura ou criação de um arquivo
  - transferência de dados
  - fechamento do arquivo
- O Java trabalha com várias classes em conjunto para facilitar a manipulação de arquivos e para que possam ser usadas é preciso importar o pacote *java.io.**.
- Em java a classe ***File*** permite representar arquivos neste nível de abstração.
- A maioria dos tratamentos de arquivos requer um bloco de exceção (*try/catch*).

Programação Orientada a Objetos

Classe ***File***

- Usada para representar o sistema de arquivos. É apenas uma abstração, a existência de um objeto *File* não significa a existência de um arquivo ou diretório.
- Contém métodos para testar a existência de arquivos, para definir permissões (nos S.O.s onde for aplicável), para apagar arquivos, criar diretórios, listar o conteúdo de diretórios, etc.

Programação Orientada a Objetos

Classe *File*

- Alguns Métodos:
  - *public String* ***getParent()***; retorna o diretório (objeto File) pai
  - *public* ***list();*** retorna lista de arquivos contidos no diretório
  - *public boolean* ***isFile()***; retorna se é um arquivo
  - *public boolean* ***isDirectory()****;* retorna se é um diretório
  - *public boolean* ***delete()****;* tenta apagar o diretório ou arquivo
  - *public long* ***length()****;* retorna o tamanho do arquivo em bytes
  - *public boolean* ***mkdir()****;* cria um diretório com o nome do arquivo
  - public String **getAbsolutePath()**; retorna o caminho absoluto (path)
  - public String ***getPath();***
  - *public String* ***getName();***

Programação Orientada a Objetos

Exemplo: Criação de Diretórios e de um Arquivo Vazio – File

```
File diretorio = new File("c:\\novo");                  /* Cria objetos com          */
diretorio.mkdir();                                                  /* referência a arquivos          */
File subdir1 = new File( diretorio, "subdir1");         /* ou diretórios e cria          */
subdir1. mkdir();                                                   /* diretórios ou subdiretorios
*/
File subdir2 = new File( diretorio, "subdir2");         /* com os nomes                */
subdir2. mkdir();                                                   /* definidos                      */
/* Cria objeto com referência ao arquivo “arquivoVazio.txt” */
File arquivo = new File( diretorio, "arquivoVazio.txt");
/* Abre o arquivo para gravação */
FileWriter f = new FileWriter(arquivo);
f.close();                                              /* fecha o arquivo */
/* Recupera a lista de arquivos do diretorio */
String[] arquivos = diretorio.list();
/* mostra o caminho absoluto de cada arquivo da lista */
for (int i =0;i<arquivos.length; i++) {
        File filho = new File( diretorio, arquivos[ i]);
        System. out. println(filho.getAbsolutePath());
}
```

Os diretórios e o arquivo serão criados:

c:\novo\subdir1
c:\novo\subdir2
c:\novo\arquivoVazio.txt

Programação Orientada a Objetos

Lendo e Escrevendo em Arquivos

- Há várias formas diferentes de ler e escrever dados:
  - sequencialmente ,aleatoriamente, como bytes, como caracteres, linha por linha, palavra por palavra…
- APIs Java para I/O oferecem objetos que abstraem fontes e destinos (nós), fluxos de bytes e caracteres e permitem que a leitura e gravação sejam feitas de diversas formas.
- São mais de 40 classes, divididas em:
  - Fluxos de entrada (*input streams*);
  - Fluxos de saída (*output streams*);
  - Leitores (*readers*);
  - Escritores (*writers*);
  - Arquivo de acesso aleatório (*random access file*).

Programação Orientada a Objetos

Lendo e Escrevendo em Arquivos

- Classes podem indicar a mídia de I/O ou a forma de manipulação dos dados e podem (devem) ser combinadas para atingirmos o resultado desejado.

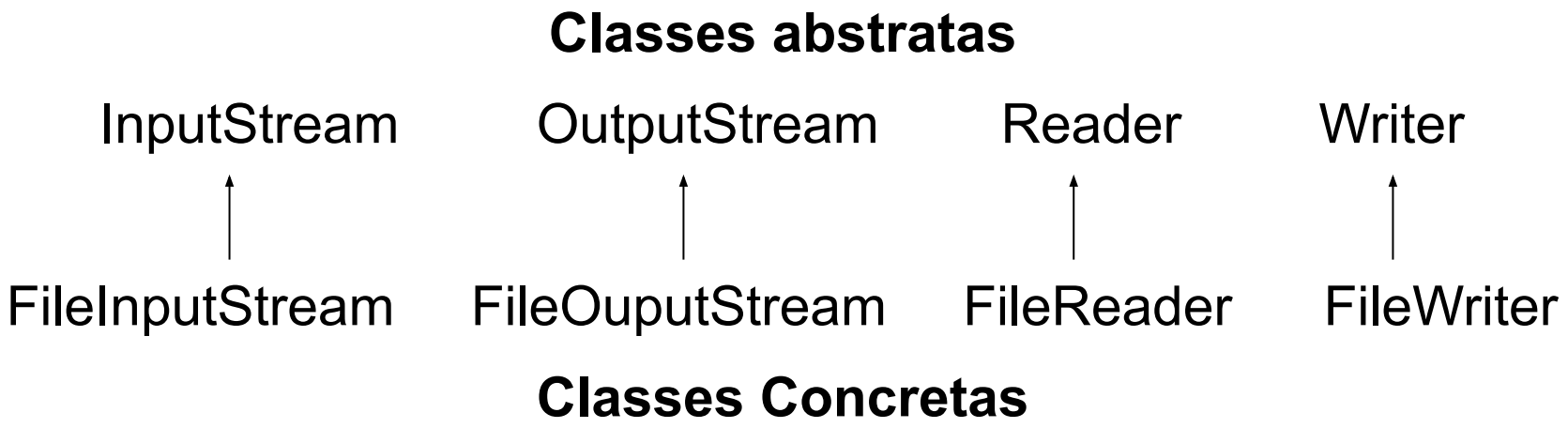
- Dois grupos:
  - Entrada e Saída de bytes:
    - InputStream e OutputStream;
  - Entrada e Saída de caracteres (chars):
    - Reader e Writer.

Programação Orientada a Objetos

Lendo e Escrevendo em Arquivos

Class *FileInputStream, FileOutputStream, FileReader, FileWriter*

- Os arquivos são abertos criando-se objetos destas classes de fluxo que herdam de InputStream, OutputStream, Reader, Writer como pode ser visto na figura.



Programação Orientada a Objetos

Leitura de byte – classe ***InputStream***

- **Métodos disponíveis nas classes Abstratas :**
  - available(), close(), read(), reset(), skip(long l),etc.

- **Classes concretas que herdam de *InputStream***
  - *FileInputStream*
  - *ObjectInputStream*
  - *AudioInputStream*
  - *Etc.*

Programação Orientada a Objetos

Leitura de caracter – classe ***Reader***

- Manipula caracteres e possui várias filhas entre elas a ***InputStreamReader*** **-** responsável pela tradução dos bytes com o encoding dado para código unicode

- Métodos disponíveis: close(), read(), ready(), reset(), skip(long l), etc.;

```
1 class TestaEntrada {
2   public static void main(String[] args) throws IOException {
3     InputStream is = new FileInputStream("arquivo.txt");
4     InputStreamReader isr = new InputStreamReader(is);
5     int c = isr.read();
6   }
7 }
```

Programação Orientada a Objetos

Leitura de String - classe ***BufferReader***

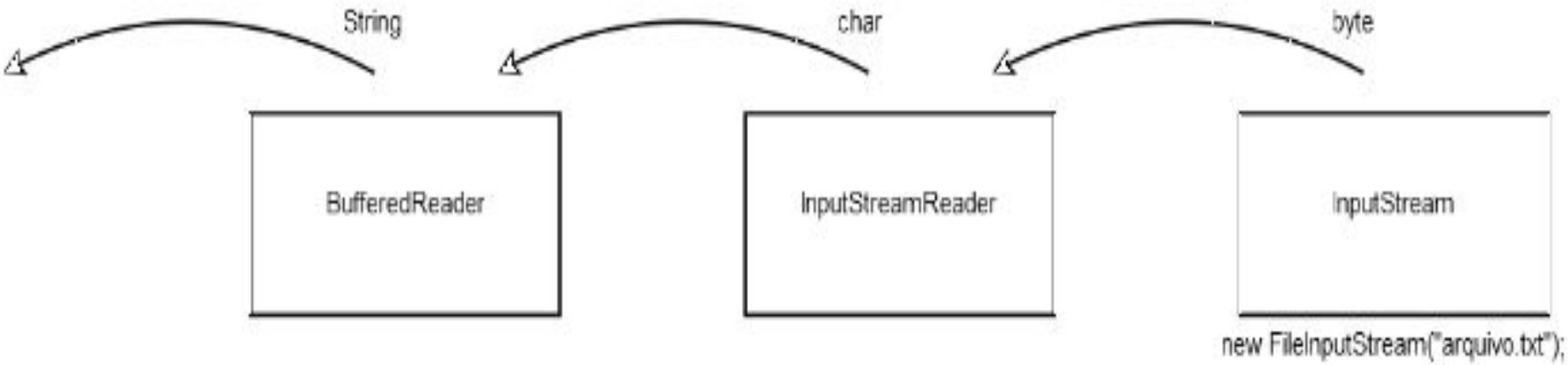
- é um *Reader* que recebe outro *Reader* pelo construtor e concatena os diversos chars para formar uma String através do método *readLine*

```
1 class TestaEntrada {
2   public static void main(String[] args) throws IOException {
3     InputStream is = new FileInputStream("arquivo.txt");
4     InputStreamReader isr = new InputStreamReader(is);
5     BufferedReader br = new BufferedReader(isr);
6     String s = br.readLine();
7   }
8 }
```

Programação Orientada a Objetos

Leitura de String - classe ***BufferReader***

- Lê o *Reader* por pedaços usando o *buffer* para evitar muitas chamadas ao sistema operacional. Composição de classes conhecida como o padrão de projetos ***Decorator Pattern*.**



Programação Orientada a Objetos

Leitura um Arquivo Inteiro

```
1  class TestaEntrada {
2    public static void main(String[] args) throws IOException {
3      InputStream is = new FileInputStream("arquivo.txt");
4      InputStreamReader isr = new InputStreamReader(is);
5      BufferedReader br = new BufferedReader(isr);
6
7      String s = br.readLine(); // primeira linha
8
9      while (s != null) {
10       System.out.println(s);
11       s = br.readLine();
12     }
13
14     br.close();
15   }
16 }
```

Programação Orientada a Objetos

Escrita de byte – classe ***OutputStream***

- **Métodos disponíveis nas classes Abstratas :**
  - close(), flush(), write(int b), write(byte[] b), etc.;

- **Classes concretas que herdam de *OutputStream***
  - *FileOutputStream*
  - *ObjectOutputStream*
  - Etc.

Programação Orientada a Objetos

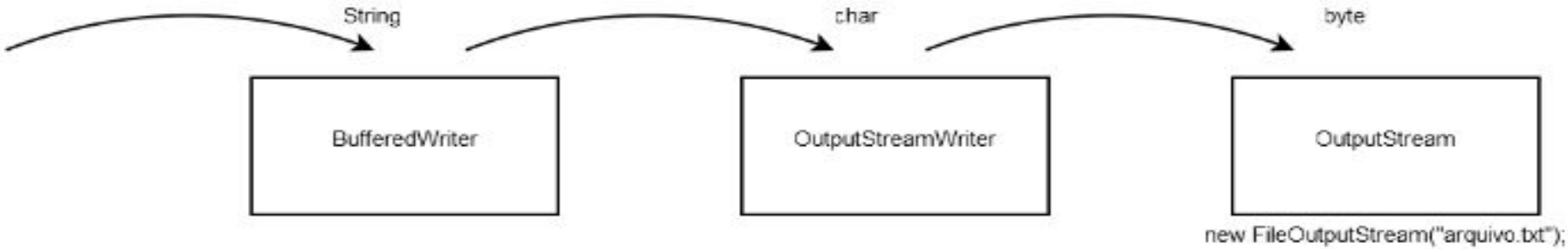
Escrita de caracter – classe ***Writer***

- **Métodos**
  - append(char c), close(), flush(), write(char[] c), write(String s), etc.

- **Classes concretas que herdam de *Writer***
  - *FileWriter*
  - *BufferedWriter*
  - *OutputStreamWriter*
  - Etc.

Programação Orientada a Objetos

Escrita em Arquivo – ***OutputStream*** *e* ***Writer***




```
class TestaSaida {
  public static void main(String[] args) throws IOException {
    OutputStream os = new FileOutputStream("saida.txt");
    OutputStreamWriter osw = new OutputStreamWriter(os);
    BufferedWriter bw = new BufferedWriter(osw);

    bw.write("teste");

    bw.close();
  }
}
```

Programação Orientada a Objetos

Exemplo: Escrita sequencial de Arquivo – *FileOutputStream*

```
File diretorio = new File("c:\\tmp");
diretorio.mkdir();
/* Cria um objeto com referência para o arquivo “lixo.txt” */
File arquivo = new File( diretorio, "lixo.txt");
/*  Cria um objeto out associando ao arquivo um fluxo de saída */
FileOutputStream out = new FileOutputStream(arquivo);
out.write( new byte[]{ 'l', 'i', 'x', 'o'} );      /* grava “lixo” no arquivo  */
File subdir = new File( diretorio, "subdir");
subdir. mkdir();                                   /* cria o diretório c:\tmp\subdir   */
String[] arquivos = diretorio.list();
for (int i =0;i<arquivos.length; i++) {            /* lista o diretório  */
     File filho = new File( diretorio, arquivos[ i]);
     System. out. println(filho.getAbsolutePath());
 }
if (arquivo.exists()) {                            /* verifica se o arquivo existe */
     arquivo.delete();                             /* e deleta   */
}
out.close();
```

Programação Orientada a Objetos

Exemplo: Escrita sequencial de Arquivo – *FileOutputStream*

```
System.out.print("Digite o texto");
/*  Cria um objeto f0 associando ao arquivo um fluxo de saída  */
FileOutputStream f0 = new FileOutputStream("c:\\Saida0.txt");
/*  Leitura de 1 byte do console padrão – teclado */
byte a = (byte)System.in.read();
while(a!='\n'){
     f0.write(a);
     a=(byte)System.in.read();
 }
  /* Arquivo Saida0.txt terá o conteúdo digitado no teclado.*/
```

Programação Orientada a Objetos

Exemplo: Leitura sequencial de Arquivo – *FileInputStream*

```
/* Cria um objeto com referência para o arquivo “c:\\listaAlunos.txt” */
File arquivo = new File("c:\\listaAlunos.txt");
/*  Cria um objeto in associando ao arquivo um fluxo de entrada */
FileInputStream in = new FileInputStream(arquivo);
/* Associa um filtro ao fluxo de entrada */
InputStreamReader conversor = new InputStreamReader(in);
/* Cria um buffer para armazenar o conteúdo lido do arquivo */
BufferedReader bf = new BufferedReader(conversor);
boolean continua=true;
String linha;
while(continua){
     linha = bf.readLine();       /* Lê linha a linha */
     if (linha==null) { continua=false; }
     else { System.out.println(linha);  }
 }
 bf.close();
 in.close();
```

Programação Orientada a Objetos

Exemplo: Leitura sequencial de Arquivo – *FileReader* e Escrita sequencial de Arquivo – *FileWriter*

- Leitura usando um *FileReader* com um *BufferedReader* – método *readLine*()
- Escrita usando um *FileWriter* com um *PrintWriter* – método *println*()
- Para facilitar a escolha do arquivo, podemos usar a classe *JFileChooser* com os métodos que mostram as janelas de diálogo (*showOpenDialog*, *showSaveDialog*, etc), *getSelectedFile* (para recuperar o arquivo selecionado) e outros atributos e métodos da classe.

Programação Orientada a Objetos

```
/* Cria objeto para utilizar a escolha de arquivos */
JFileChooser fc = new JFileChooser();
/* Mostra a janela de dialogo para escolha do arquivo */
result = fc.showOpenDialog(null);
/* Verifica se foi selecionada a opção Cancela da janela */
if (result == JFileChooser.CANCEL_OPTION)
    System.exit(0);
else {
    /* Cria objeto com referência para o arquivo físico selecionado */
    File farqEnt = fc.getSelectedFile();
    try {
        /* Abre o arquivo para Leitura */
        FileReader f = new FileReader(farqEnt);
        /* Classe BufferedReader permite a leitura linha a linha */
        BufferedReader in = new BufferedReader(f);
        /* Lê linha a linha e mostra na console */
        String linha = in.readLine();
        while(linha !=null ){
            System.out.println(linha);
            linha = in.readLine();
         }
         in.close();
     } catch (Exception e) { System.out.println(e); }
}
```

Programação Orientada a Objetos

Exemplo: Gravação sequencial de Arquivo – *FileWriter*

```
/* Cria com a referência para o arquivo físico de gravação */
FileWriter fw = new FileWriter("arquivo.txt");
/* Abre um fluxo associado ao arquivo */
PrintWriter pw = new PrintWriter(fw);

/* para cada linha digitada, grava a linha no arquivo */
String linha = JOptionPane.showInputDialog("linha:");
while(!linha.equals("fim") ){
     pw.println(linha);        /* Escreve no arquivo */
     pw.flush();             /* Confirma a gravação*/
     linha = JOptionPane.showInputDialog("linha:");
 }
 pw.close();                 /* fecha o fluxo de impressão */
```

Programação Orientada a Objetos

Arquivos (cont..)

- *FileInputStream* e *FileOutputStream* são adequadas para escrever dados binários (*bytes*).
- Para manipular mais facilmente texto (*strings*) podemos utilizar as classes *FileReader*, *BufferedReader*, *FileWriter* e *PrintWriter*.

- A seguir, dois exemplos de cópia de arquivos

Programação Orientada a Objetos

Exemplo de Cópia de Arquivos (bytes)

```
import java.io.*;
public class CopyBytes {
    public static void main(String[] args) throws IOException {
        File inputFile = new File("tst-input.txt");
        File outputFile = new File("tst-output.txt");
        FileInputStream in = new FileInputStream(inputFile);
        FileOutputStream out = new FileOutputStream(outputFile);
        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
        }
        in.close();
        out.close();
    }
}
```

Programação Orientada a Objetos

Exemplo de Cópia de Arquivos (linha a linha)

```
import java.io.*;
public class CopyBytes2 {
        public static void main(String[] args) throws IOException {
                FileReader inputFile = new FileReader("oi.txt");
                FileWriter outputFile = new FileWriter("ola.txt");
                BufferedReader in = new BufferedReader(inputFile);
                PrintWriter out = new PrintWriter(outputFile);
                String s;
                while ((s = in.readLine()) != null)
                        out.println(s);
                        out.flush();
                 }
                in.close();
                out.close();
        }
}
```

Programação Orientada a Objetos

Considerações

- Note que é necessário executar o método ***flush*** nos objeto de escrita para que os dados sejam efetivamente escritos no arquivo de saída (limpar o buffer).

- A diferença principal com relação ao exemplo anterior é que neste programa fazemos a leitura e escrita de linhas de texto (métodos *readLine* e *println*).

Programação Orientada a Objetos

Considerações

- Para imprimir *strings* na saída padrão utilizamos os métodos *print* ou *println*.

- Para ler algum valor do teclado, devemos criar um objeto da classe *BufferedReader* que possui o método *readLine* utilizado anteriormente.

- BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));
- String linha = teclado.readLine();

Programação Orientada a Objetos

Exercício:

- Criar um método que leia um arquivo byte a byte, mostre na tela o resultado e grave em outro arquivo (cópia);

- Criar um método que leia o mesmo arquivo linha a linha e mostre na tela o resultado (cópia);

Obs: Escolha um arquivo existente no seu sistema de arquivos e execute o programa com esse arquivo.